An Approach for Modeling Polyglot Persistence

Cristofer Zdepski, Tarcizio Alexandre Bini and Simone Nasser Matos
Federal Technological University of Parana, Av Monteiro Lobato, s/n - Km 04, 84016210, Ponta Grossa, Parana, Brazil

Keywords: NoSQL, Database Design, Polyglot Persistence, Logical Level Design.

Abstract: The emergence of NoSQL databases has greatly expanded database systems in both storage capacity and performance. To make use of these capabilities many systems have integrated these new data models into existing applications, making use of multiple databases at the same time, forming a concept called ”*Polyglot Persistence*”. However, the lack of a methodology capable of unifying the design of these integrated data models makes design a difficult task. To overcome this lack, this paper proposes a modeling methodology capable of unifying design patterns for these integrated databases, bringing an overview of the system, as well as a detailed view of each database design.

1 INTRODUCTION

The popularization of Web 2.0 brought an dramatically increase in data generation. Nowadays databases easily exceed petabytes scale, with daily increase rates in the terabytes scale. As an example we can take the Facebook database, which in 2014 had a size of 300 petabytes, and a daily increase rate of 600 terabytes, according to (Wilfong and Vagata, 2014). Thus the recent changes, mainly caused by Web 2.0, at the level of users, infrastructure and applications characteristics has transformed the use of relational model an increasingly difficult task.

Relational databases have been the market leaders for the past 40 years due to their great ability to adapt to most problems. The long time of existence also gives it a high level of maturity, and is still the most recommended for many applications, mainly the ones that need a high level of consistency on data manipulation and storage. One of the main characteristics of relational databases is the implementation of ACID properties (**A**tomicity, **C**onsistency, **I**solation and **D**urability). These properties ensures a strong consistency and guarantees validity of data even in the event of errors, like power or network failures.

Despite meeting the needs of many different current applications, the relational model begins to show problems when dealing with very large databases, mainly with performance problems. The main reason is in fact that they were not designed to be scaled, especially when it comes to horizontal scalability. The inherent need for dynamic schema of Web 2.0 applications also brings out some weaknesses of the relational model by working with static schemas.

Inspired by this limitations, NoSQL databases have emerged as the solution to the growing need of systems for information. As they expand over the Internet its data increases even faster. According to (Sadalage and Fowler, 2012) and (Cattell, 2011) the biggest goal of NoSQL databases is their scalability skills that are required for next generation web applications. While relational databases rely on high consistency, provided by ACID properties, many NoSQL databases works with BASE (**B**asically **A**vailable, **S**oft state, **E**ventually consistent) properties which aims to provide high levels of availability and resilience even though this may compromise consistency for a few moments.

There are many different NoSQL databases nowadays and their data structures vary from one another. According to (Sadalage and Fowler, 2012; Hashem and Ranc, 2016; Kaur and Rani, 2013; Abramova et al., 2014) NoSQL databases can be classified into the following data models, grouped by their main characteristics: (1) *Key-value*; (2) *Document-Oriented*; (3) *Column-family*; (4) *Graph databases*. Even among the different categories, the concept of *”schema-less”* is widely used in the NoSQL databases. Many of them have no schema definition or data constraints. Even object creation that is mandatory in relational databases is often not required in NoSQL databases as they are created when used.

Database design spends a great deal of time on developing conceptual, logical, and physical models.

Zdepski, C., Bini, T. and Matos, S.
An Approach for Modeling Polyglot Persistence.
DOI: 10.5220/0006684901200126
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 120-126
ISBN: 978-989-758-298-1

The main purpose of these models is to ensure that the structures needed to store the data will be properly constructed and will meet the system requirements. Even so, database design is not commonly applied to NoSQL databases. The so-called "schema-less" in NoSQL databases may give the impression of an attempt to eliminate the need for data modeling, but the modeling aims more than just the schema definition. Database modeling mainly seeks a better understanding and easier visualization of the data structures to be used.

(Sadalage and Fowler, 2012) mentions that the allegation of absence of schema in NoSQL databases is misleading. Even if the database does not consider a schema associated with the data being stored, such a schema must be defined by the application because data needs to be understood by the application in order to be processed or stored. If the application fails to parse the data, we have a schema mismatch and the application could not function. Therefore, even in *"schema-less"* NoSQL databases, it is important to have a suitable design so that the data can be properly stored and processed.

The correct use of NoSQL databases have to be always considered. Despite its current popularity, many applications do not justify the use of a NoSQL database and, if done, can lead to more losses than gains. Some systems will require the use of a NoSQL data model only for part of their data, while the other parts may be more suitable for another NoSQL or even a relational data model.

Complex applications can take better advantage of using different data models for storing and processing different parts of their data. An e-commerce system could, for example, be implemented according to Figure 1. This would bring distinct structures to system parts, making the best use of the capabilities of each data model. Still, an unified view would simplify the visualization of interactions between the parts, allowing an overview of the system as a whole.

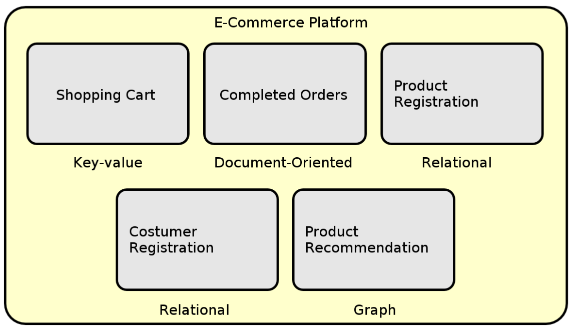

Figure 1: Suggested E-Commerce Platform Database.

These kind of complex applications are the object of study of this paper. It explores a proposal for modeling systems based on what we can call a *"Polyglot Database System"*, which uses multiple different data models, each one with their own particular objectives. This paper aims to propose a modeling methodology capable of integrating multiple different data models into a single system, making the best of each one and maintaining an unified view of the system as a whole.

This paper is organized as follows. Section 2 presents some fundamental concepts of database design and related works. Section 3 presents the suggested modeling proposal and finally, Section 4 concludes and presents future works.

2 BACKGROUND AND RELATED WORKS

The absence of design methodologies for NoSQL databases is a topic already explored by several authors who proposed different solutions. Many of these solutions are limited to specific data models or steps in database design and none attempts to explore a methodology that can standardize the complete design of databases for emerging data models in conjunction with existing ones. The combination of multiple databases with different data models in a single system, called by (Sadalage and Fowler, 2012) of "*Polyglot Persistence*" is also a topic barely explored in database design.

It is a consensus among several authors such as (Rob and Coronel, 2007; Ramakrishnan and Gehrke, 2000; Martyn, 2000) that the database design is essentially divided into 3 steps: (1) *Conceptual Design*, (2) *Logical Design* and (3) *Physical Design*. Conceptual design aims to translate the requirements into a conceptual database schema. According to (Korth and Silberschatz, 1986), the model developed at this step provides a detailed overview of the whole system context. It is well known that the conceptual design is *technology-independent* and for this reason it can be applied to both relational database and NoSQL.

Unlike the conceptual design, the logical design brings the conceptual model previously developed to a step much more dependent on the adopted data model. According to (Rob and Coronel, 2007), the logical design realizes the translation of the conceptual model for the internal model of a DBMS (Database Management System) and for this reason it is dependent on data model to be employed. Many of the specificities of each database should be explored, such as the supported data types and the format the data will be stored, such as tables, documents or nodes and edges.

In the physical design step physical peculiarities of each database software are treated. These peculiarities involve specific types of data, forms of storage, partitioning, clustering capabilities among others. (Martyn, 2000) mentions that the main purpose of physical design is to make the data model machine efficient. The physical definition is necessary to ensure a better match of the data to the storage software. In the same way that the schema definition is necessary for the application to function properly, the physical definition is necessary to ensure the efficiency of the database.

The challenge for modeling NoSQL databases begins with conversion from the conceptual to logical model. (Banerjee and Sarkar, 2016) suggests the definition of a rule-based conversion system capable of transforming the conceptual model into logical and later into a physical model compatible with the four categories of NoSQL databases. However, the model does not explore the possibility of using multiple databases and therefore does not define the boundaries between them.

The NoSQL Abstract Model (NoAM) is proposed by (Bugiotti et al., 2013) and explored later by (Bugiotti et al., 2014) defining a methodology capable of creating a model for the aggregate-based databases. Its methodology proposes that the same model can be used for the elaboration of the three data models based on aggregate and the definition of which to use can be done in the physical design step. This could be a problem due to capabilities of each data model can change the logical modeling, such as how to obtain the data. As an example, key-value databases generally do not have data searching capabilities, and this could affect the logical design of the database. So defining the data model after the logical design can lead to erroneous definitions that would need to be subsequently adjusted.

A very relevant point in a design involving NoSQL databases is the way of querying the data. Due to many of these databases do not have complex query structures like joins, they need to be designed for greater query efficiency and simplicity, as described in (Li et al., 2014). In a context with multiple databases a concern with query should be even greater, since a query that involves several databases can bring even more complexity to the process.

A final concern is the graphical notation capable of representing the new data structures created by these databases. While relational databases work only with tables and rows, NoSQL databases bring aggregates or graph structures and their representation clearly in existing logical models can be a difficult task. In order to solve this deficiency, (Jovanovic and Benson, 2013) suggests a modeling style that uses IDEF1X to represent the aggregate structures. But graph structures still can not be represented by this modeling style.

We can therefore note that several efforts exist in order to elaborate processes capable of assisting in the design of these NoSQL databases, but generally are isolated efforts in specific areas. The absence of a model capable of unifying the whole process, joining several data models when necessary and exploring the best characteristics of each one, motivated us to elaborate our proposal.

3 PROPOSAL

Complex database systems can usually be divided into subsystems that are quite different from each other. Some subsystems have needs that would be better met by NoSQL databases due to their need for scalability, availability and performance. However, other subsystems have needs that would best be met by classic relational databases because of their consistency and atomicity. Often a single database system, being relational or NoSQL, would not be able to satisfy both the consistency and atomicity scenario and the scalability, availability and performance required by these complex database systems given the wide range of concepts between them. Taking this as a premise, many of these systems can best be implemented by using multiple integrated databases, in order to extract the best usage for each one. Our proposal aims to create a methodology for designing polyglot database systems, allowing from an overview of the entire system to its specific details.

The main difference between our proposal and the others presented is the subdivision of the conceptual model into subsystems. These units represent the possible divisions between the multiple data models required to implement the database system. Part of the modeling has to be the definition of the target data model, checking their requirements to find the best option for that subsystem. The basis of this methodology is the classic database design as cited in section 2 and it follows the three stages defined therein. Our methodology subdivides the steps of the database design in order to allow the segmentation of the model in subsystems, as shown in Figure 2.

As the conceptual model is *technology-independent*, we propose the use of the entity-relationship model (ER). It is a widely known and exploited model and allows a detailed definition of the entities and their relationships on the database. Our proposal does not intend to modify or extend this step of database design. The proposed modifications

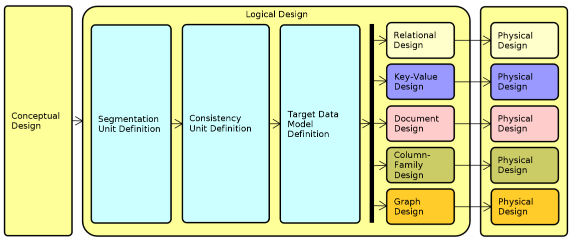

Figure 2: Proposed Design Steps.

begin with the logical design step, as described in the following sections.

3.1 Segmentation Unit Definition

As the definition of a polyglot database system, multiple databases with multiple data models are used on their conception. Each data model has to be allocated to parts of the system where it better suits and taking full advantage of its features. To make this possible, is necessary to define the borders between each subsystem, defining what we will call by *segmentation unit*.

Segmentation in this context consists on divide the conceptual model in parts according to main features to be implemented. Main goal here is to define the subsystems that is fully functional units and independent of other units. This does not means that units cannot relate to each others, but that a unit cannot depend on this relation to be functional. To help us understand the concept, let us consider a simplified e-commerce system as illustrated in Figure 3. It illustrates an e-commerce system, consisting of five subsystems: (1) *Shopping Cart*, (2) *Costumer Registration*, (3) *Product Registration*, (4) *Completed Orders* and (5) *Product Recommendation*. The entities for each unit are surrounded by a rounded rectangle. These units share some entities such as *Product*, but all can operate independently.

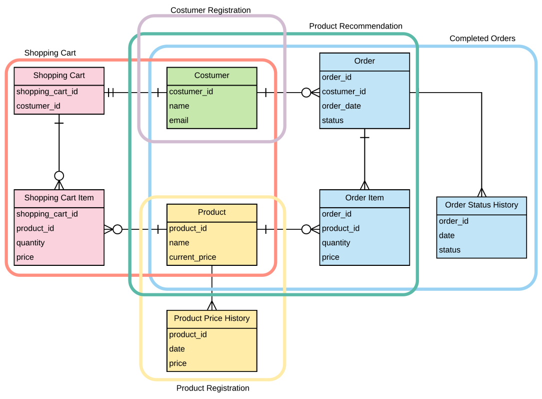

Figure 3: E-Commerce System Segmentation Units.

Sometimes an entity may be shared by two or more segmentation units. Examples can be seen in Figure 3 such as the entity *Product*. The *Shopping Cart*, *Completed Orders* and *Product Recommendation* shares an entity that is owned by the *Product Registration* subsystem, the *Product*. When that happens, the entities must be replicated between units, with necessary attributes for each segment. There is no need for replicas to be exactly equals, but they need an identification that allows them to be interrelated and synchronized between units. This is also necessary to maintain the functional independence of the unit.

The main reason for definition of *segmentation units* is the capability of use multiple different data models at the same system. Segmentation units define cut points on the system that can be used to split on multiple different data models, as necessary. At this point, we have not yet settled the target database, just the cut points that will not corrupt the conceptual model, making possible the division. Once this is done, it is possible to detach any of the segmentation units from the model, and work as an independent system.

This definition does not necessarily means that we will use different data models on implementation. As multiple segmentation units can share the same data model, there is no reason for the implementation on multiple databases in this case.

3.2 Consistency Unit Definition

When working with NoSQL databases, one of the most common features is the so-called eventual consistency, provided by BASE properties. This raises concerns about the consistency of data sets. While data does not need to be consistent across the entire database, some data groups need to be consistent to be considered valid. These groups are what we called by *Consistency Units*. The definition of such a unit also allows data fragmentation, common practice to ensure horizontal scalability. In the context of aggregate-oriented databases, consistency units form the aggregates themselves, since they are the atomic operation unit of the database.

A practical example is the *Completed Orders* subsystem shown in Figure 4. The order data as well as the *order items*, *product* and *costumer* must compose a consistency unit since they must always be consistent with each other or the order data will be invalid. If for the eventual consistency property, in some situation a query returns the order without all their items, we have an inconsistency that may cause a system failure. So they have to be fully consistent. The same does not happen with the *Order Status History* entity. As this a historical query entity, an eventual inconsistency will not affect the system. As the model in Figure 3 is a simplified example, other segmentation units are formed by only one unit of consistency.

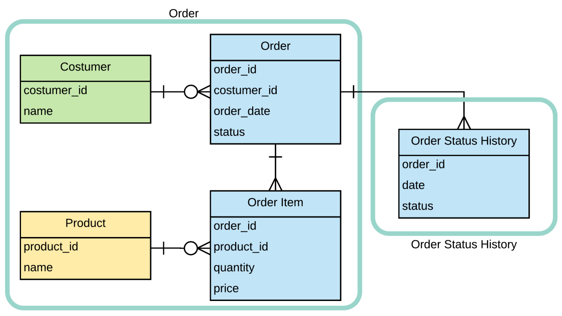

Figure 4: Completed Orders Consistency Units.

This consistency concern does not exist with ACID databases because it is guaranteed for the entire database, but at this point we do not yet have the definition of which data model will be applied to that segmentation unit. The definition of these consistency units will aid in this task by defining the inseparable data units and consistency constraints that may in some cases lead to change the selected data model.

The definition of *consistency units* is also affected by the way the data is queried. Since some NoSQL databases do not have complex query structures, such as joins, data must also be designed to be extracted in the best way. Many of these databases are only capable of extracting data from a complete consistency unit at a time, such as key-value databases. Therefore, in our case, including historical status data within the order can increase the amount of unnecessary data received by the application, as the historical data is not always used.

3.3 Target Data Model Definition

With proper knowledge about the subsystems and their consistency needs, the next step is to define the best data model to be adopted for each segmentation unit. This can be the classic relational model or any of the NoSQL database categories described. The definition requires prior knowledge of features from the data models to tailor the data to the specified performance, scalability and consistency requirements. The important thing is to identify which data model better suits the needs for each subsystem.

To help a better understanding of target data model definition, let's again consider the segmentation units defined in Figure 3. Since each of the segmentation units has its own characteristics, it is necessary to evaluate them individually to determine the best data model to be adopted.

At the *Shopping Cart* subsystem, we have only one unit of consistency, that is the cart with products and quantities selected by the customer. The *product* does not belong directly to this unit, as the *costumer*, but they are referenced by it. An important requirement to consider in this subsystem is performance and availability. The unavailability of this system or slowness could cause the loss of a potential customer. Eventual consistency would also be acceptable as it would not cause system losses. Therefore, a NoSQL database would be a good candidate. An important thing to be considered is the need for searches. Because the shopping cart is always customer-related, it can be said that no searches will be required to find the *shopping cart*. These two factors justify the implementation of this subsystem with a key-value data model.

The subsystem of *Product Registration* is responsible for the products and their prices. As it is a simplified view, we also have only one consistency unit, composed by the *product* and its *price history*. In this case, searches may be required to find products through their attributes such as name or price. Availability is also a vital factor as data is accessed by other subsystems, including the *Shopping Cart* and *Product Registration*. Consistency also needs a little more consideration because product prices are being handled, but we can still think about eventual consistency. For this scenario, we can consider a document-oriented model as a good candidate, since it allows searches and also high availability. A column-family can also be used.

In the *Completed Order* and *Costumer Registration* subsystems, consistency is the main factor to be considered. Availability is always a relevant factor but in this case consistency is preferred. Once you place an order, your values and quantities need to be 100% consistent, to avoid problems with payment integrations. For this reason, a data model with ACID properties would fit well, so a relational data model can be used.

The last of the subsystems is the *Product Recommendation*. This subsystem is based on searches on the latest purchases related to *customers* and *products*, usually with relationships such as "Customers who bought the product X also bought the product Y". This type of search is related to navigation between purchases and customers, in a structure like a graph. Consistency here is not so necessary, as it is a recommendations system only and an eventual inconsistency will not cause any losses. By performance characteristics and query model, a graph data model is recommended. They are especially designed for this kind of situation. As graph databases are normally

ACID compliant, consistency will not be a problem.

As we can see, the definition of the data model that is most appropriate to each subsystem requires a knowledge of the subsystem itself and the data models, in order to discover the most relevant characteristics in each case. Our example is merely illustrative and a more detailed scenario may lead to decisions different from those described here.

3.4 Logical Data Model Design

The last step in the logical design of our proposal deals with the individual logical design of each of the previously selected data models. It is beyond the scope of this paper the detailing of this process that must be individualized for each of the data models to be adopted. The goal here is to outline the objectives to be achieved with this process.

As the objective of the conceptual design step is the translation of the conceptual model into the internal model of a DBMS (Rob and Coronel, 2007) the previous steps just prepared the template for this translation, defining the target data model and the consistency units. This translation consists in transform the consistency units and its entities and attributes into the data structures of the target data model. On a document-oriented data model, this means transform the consistency unit in an aggregate, as this is the atomic unit of this data model, and define its collection. On a key-value data model, the consistency unit is also the aggregate, but the key have to be defined.

The definitions of each of the data models must be treated in a specific way due to the particularities of each one. Even between aggregate-oriented databases, features such as search capabilities, data storage format, and others can determine essential differences in data models.

4 CONCLUSION AND FUTURE WORKS

NoSQL databases have emerged to improve the software capabilities of storage and performance, providing ways to work with the so-called "*Big Data*". However, the use of these data models is still largely based on best practices and examples and there are few initiatives to standardize the documentation and methodologies for modeling such databases. Most of the related works presented on this paper are based on specific data models, notations or starts an entirely new modeling process, without taking advantage of existing knowledge about database design.

Our solution aims to bring a design standardization, providing a unified methodology capable of working with several data models integrated into a single system. The concept is to extend the existing database design by adding the steps required to model complex systems with multiple integrated data models. To explore this methodology we have described a simplified example, but with definitions compatible with a complex system.

This work is the initial phase of a larger work seeking for a complete modeling strategy for database systems that use the so-called "*Polyglot Persistence*". Future works can explore the logical and physical design steps of each of existing data models, such as aggregate-oriented and graph data models. A graphical notation for represent the segmentation and consistency units is also a necessity for bringing more visual understanding to the design diagrams. This need for a graphical notation is also a necessity for the logical design of the data models, as described by (Jovanovic and Benson, 2013) about aggregate-oriented data models, but also applies to graphs.

REFERENCES

Abramova, V., Bernardino, J., and Furtado, P. (2014). Which nosql database? a performance overview. *Open Journal of Databases (OJDB)*, 1(2):17–24.

Banerjee, S. and Sarkar, A. (2016). Logical level design of nosql databases. In *2016 IEEE Region 10 Conference (TENCON)*, pages 2360–2365.

Bugiotti, F., Cabibbo, L., Atzeni, P., and Torlone, R. (2013). A logical approach to nosql databases.

Bugiotti, F., Cabibbo, L., Atzeni, P., and Torlone, R. (2014). Database design for nosql systems. In *International Conference on Conceptual Modeling*, pages 223–231. Springer.

Cattell, R. (2011). Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27.

Hashem, H. and Ranc, D. (2016). Evaluating nosql document oriented data model. In *Future Internet of Things and Cloud Workshops (FiCloudW), IEEE International Conference on*, pages 51–56. IEEE.

Jovanovic, V. and Benson, S. (2013). Aggregate data modeling style. *SAIS 2013*, pages 70–75.

Kaur, K. and Rani, R. (2013). Modeling and querying data in nosql databases. In *2013 IEEE International Conference on Big Data*, pages 1–7.

Korth, H. F. and Silberschatz, A. (1986). *Database System Concepts*. McGraw-Hill, Inc., New York, NY, USA.

Li, X., Ma, Z., and Chen, H. (2014). Qodm: A query-oriented data modeling approach for nosql databases. In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pages 338–345.

Martyn, T. (2000). Implementation design for databases: the ‘forgotten’ step. *IT Professional*, 2(2):42–49.

Ramakrishnan, R. and Gehrke, J. (2000). *Database management systems*. McGraw Hill.

Rob, P. and Coronel, C. (2007). *Database Systems: Design, Implementation, and Management*. Course Technology Press, Boston, MA, United States, 8th edition.

Sadalage, P. J. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition.

Wilfong, K. and Vagata, P. (2014). Scaling the facebook data warehouse to 300 pb. Accessed: 2017-09-10.